The Chess Stress Test for Discrete Choice Modeling

Kenneth W. Regan
University at Buffalo (SUNY)

UB CSE UpBeat, 10/19/2018

\[1\] Joint work with Tamal Tanu Biswas and with grateful acknowledgment to UB’s Center for Computational Research (CCR)
Multinomial Logit Model

Given options $m_1, \ldots, m_J$ and information $X = X_1, \ldots, X_J$ about all of them, and characteristics $S$ of a person choosing among them, we want to project the probabilities $p_j$ of $m_j$ being chosen.
Multinomial Logit Model

Given options $m_1, \ldots, m_J$ and information $X = X_1, \ldots, X_J$ about all of them, and characteristics $S$ of a person choosing among them, we want to project the probabilities $p_j$ of $m_j$ being chosen. First define numbers $u_j = g(X, S)_j$ often thought of as “utilities.”
Multinomial Logit Model

Given options $m_1, \ldots, m_J$ and information $X = X_1, \ldots, X_J$ about all of them, and characteristics $S$ of a person choosing among them, we want to project the probabilities $p_j$ of $m_j$ being chosen. First define numbers $u_j = g(X, S)_j$ often thought of as “utilities.” Then the \textit{multinomial logit} (MNL) model represents the probabilities via

$$\log(p_j) = \alpha + \beta u_j.$$
Given options $m_1, \ldots, m_J$ and information $X = X_1, \ldots, X_J$ about all of them, and characteristics $S$ of a person choosing among them, we want to project the probabilities $p_j$ of $m_j$ being chosen. First define numbers $u_j = g(X, S)_j$ often thought of as “utilities.” Then the *multinomial logit* (MNL) model represents the probabilities via

$$\log(p_j) = \alpha + \beta u_j.$$ 

The quantities

$$L_j = e^{\alpha + \beta u_j}$$

are called *likelihoods*. 
Multinomial Logit Model

Given options $m_1, \ldots, m_J$ and information $X = X_1, \ldots, X_J$ about all of them, and characteristics $S$ of a person choosing among them, we want to project the probabilities $p_j$ of $m_j$ being chosen. First define numbers $u_j = g(X, S)_j$ often thought of as “utilities.” Then the *multinomial logit* (MNL) model represents the probabilities via

$$\log(p_j) = \alpha + \beta u_j.$$ 

The quantities

$$L_j = e^{\alpha + \beta u_j}$$

are called *likelihoods*. Then the probabilities are obtained simply by normalizing them:

$$p_j = \frac{L_j}{\sum_{j'=1}^J L_{j'}} = \text{def } \text{softmax}(\beta u_1, \ldots, \beta u_J).$$
Multinomial Logit Model

Given options $m_1, \ldots, m_J$ and information $X = X_1, \ldots, X_J$ about all of them, and characteristics $S$ of a person choosing among them, we want to project the probabilities $p_j$ of $m_j$ being chosen. First define numbers $u_j = g(X, S)_j$ often thought of as “utilities.” Then the multinomial logit (MNL) model represents the probabilities via

$$\log(p_j) = \alpha + \beta u_j.$$ 

The quantities

$$L_j = e^{\alpha + \beta u_j}$$

are called likelihoods. Then the probabilities are obtained simply by normalizing them:

$$p_j = \frac{L_j}{\sum_{j'=1}^J L_{j'}} = \text{def softmax}(\beta u_1, \ldots, \beta u_J).$$

Finally obtain $\beta$ by fitting; $e^\alpha$ becomes a constant of proportionality so that the $p_j$ sum to 1.
The Chess Stress Test for Discrete Choice Modeling

Chess Decision Setting

One player $P$ with characteristics $S$. Multiple game turns, each has possible moves $m_t$; $j$. For a given turn (i.e., chess position) $t$, legal moves are $m_1; \ldots ; m_j; \ldots ; m_J$ (index $t$ understood).

Moves indexed by values $v_1; \ldots ; v_J$ in nonincreasing order. Values determined by strong chess programs. Not apprehended fully by $P$ (bounded rationality, fallible agents).

Raw utilities $u_j = (v_1; v_j)$ by some difference-in-value function in either pawn units or chance of winning units. Parameter treated as a divisor $s$ of those units, i.e., $\frac{1}{s}$.

Second parameter $c$ allows nonlinearity: $(v_1; v_i)^c$. (First $c = 1$.)

MNL model (called Shares by me) then equivalent to:

$$\log(p_j) = U_j = \left(\frac{v_1}{v_j}\right)^s c \text{ and we go as before.}$$

Taking $\log(p_j) = \log(p_1)$ on LHS gives same model.
Chess Decision Setting

- One player $P$ with characteristics $S$. 
Chess Decision Setting

- One player $P$ with characteristics $S$.
- Multiple *game turns* $t$, each has possible moves $m_{t,j}$.
Chess Decision Setting

- One player $P$ with characteristics $S$.
- Multiple *game turns* $t$, each has possible moves $m_{t,j}$.
- For a given turn (i.e., chess position) $t$, legal moves are $m_1, \ldots, m_j, \ldots, m_J$ (index $t$ understood).
Chess Decision Setting

- One player $P$ with characteristics $S$.
- Multiple *game turns* $t$, each has possible moves $m_{t,j}$.
- For a given turn (i.e., chess position) $t$, legal moves are $m_1, \ldots, m_j, \ldots, m_J$ (index $t$ understood).
- Moves indexed by values $v_1, \ldots, v_J$ in nonincreasing order.
Chess Decision Setting

- One player $P$ with characteristics $S$.
- Multiple game turns $t$, each has possible moves $m_{t,j}$.
- For a given turn (i.e., chess position) $t$, legal moves are $m_1, \ldots, m_j, \ldots, m_J$ (index $t$ understood).
- Moves indexed by values $v_1, \ldots, v_J$ in nonincreasing order.
- Values determined by strong chess programs. Not apprehended fully by $P$ (bounded rationality, fallible agents).
Chess Decision Setting

- One player $P$ with characteristics $S$.
- Multiple game turns $t$, each has possible moves $m_{t,j}$.
- For a given turn (i.e., chess position) $t$, legal moves are $m_1, \ldots, m_j, \ldots, m_J$ (index $t$ understood).
- Moves indexed by values $v_1, \ldots, v_j$ in nonincreasing order.
- Values determined by strong chess programs. Not apprehended fully by $P$ (bounded rationality, fallible agents).
- Raw utilities $u_j = \delta(v_1, v_j)$ by some difference-in-value function $\delta$ in either “pawn units” or “chance of winning” units.
Chess Decision Setting

- One player $P$ with characteristics $S$.
- Multiple *game turns* $t$, each has possible moves $m_{t,j}$.
- For a given turn (i.e., chess position) $t$, legal moves are $m_1, \ldots, m_j, \ldots, m_J$ (index $t$ understood).
- Moves indexed by values $v_1, \ldots, v_J$ in nonincreasing order.
- Values determined by strong chess programs. Not apprehended fully by $P$ (*bounded rationality, fallible agents*).
- Raw utilities $u_j = \delta(v_1, v_j)$ by some difference-in-value function $\delta$ in either “pawn units” or “chance of winning” units.
- Parameter $\beta$ treated as a divisor $s$ of those units, i.e., $\beta = \frac{1}{s}$.
Chess Decision Setting

- One player $P$ with characteristics $S$.
- Multiple *game turns* $t$, each has possible moves $m_{t,j}$.
- For a given turn (i.e., chess position) $t$, legal moves are $m_1, \ldots, m_j, \ldots, m_J$ (index $t$ understood).
- Moves indexed by values $v_1, \ldots, v_J$ in nonincreasing order.
- Values determined by strong chess programs. Not apprehended fully by $P$ (*bounded rationality, fallible agents*).
- Raw utilities $u_j = \delta(v_1, v_j)$ by some difference-in-value function $\delta$ in either “pawn units” or “chance of winning” units.
- Parameter $\beta$ treated as a divisor $s$ of those units, i.e., $\beta = \frac{1}{s}$.
- Second parameter $c$ allows nonlinearity: $\delta(v_1, v_i)^c$. (First $c = 1$.)
Chess Decision Setting

- One player $P$ with characteristics $S$.
- Multiple *game turns* $t$, each has possible moves $m_{t,j}$.
- For a given turn (i.e., chess position) $t$, legal moves are $m_1, \ldots, m_j, \ldots, m_J$ (index $t$ understood).
- Moves indexed by values $v_1, \ldots, v_J$ in nonincreasing order.
- Values determined by strong chess programs. Not apprehended fully by $P$ (*bounded rationality, fallible agents*).
- Raw utilities $u_j = \delta(v_1, v_j)$ by some difference-in-value function $\delta$ in either “pawn units” or “chance of winning” units.
- Parameter $\beta$ treated as a divisor $s$ of those units, i.e., $\beta = \frac{1}{s}$.
- Second parameter $c$ allows nonlinearity: $\delta(v_1, v_i)^c$. (First $c = 1$.)
- MNL model (called “Shares” by me) then equivalent to:

$$\log(p_j) = U_j = \left(\frac{\delta(v_1, v_j)}{s}\right)^c$$

and we go as before.
Chess Decision Setting

- One player $P$ with characteristics $S$.
- Multiple *game turns* $t$, each has possible moves $m_{t,j}$.
- For a given turn (i.e., chess position) $t$, legal moves are $m_1, \ldots, m_j, \ldots, m_J$ (index $t$ understood).
- Moves indexed by values $v_1, \ldots, v_J$ in nonincreasing order.
- Values determined by strong chess programs. Not apprehended fully by $P$ (*bounded rationality, fallible agents*).
- Raw utilities $u_j = \delta(v_1, v_j)$ by some difference-in-value function $\delta$ in either “pawn units” or “chance of winning” units.
- Parameter $\beta$ treated as a divisor $s$ of those units, i.e., $\beta = \frac{1}{s}$.
- Second parameter $c$ allows nonlinearity: $\delta(v_1, v_i)^c$. (First $c = 1$.)
- MNL model (called “Shares” by me) then equivalent to:

$$\log(p_j) = U_j = \left(\frac{\delta(v_1, v_j)}{s}\right)^c$$

and we go as before. Taking $\log(p_j) - \log(p_1)$ on LHS gives same model.
Alternative “Loglog-Linear” Model

Represent a difference in \textit{double logs} of probabilities on left-hand side instead.
Alternative “Loglog-Linear” Model

Represent a difference in *double logs* of probabilities on left-hand side instead. Now nice to keep signs nonnegative by inverting probabilities.

\[
\log \log(1/p_j) - \log \log(1/p_1) = \beta U_j
\]
Alternative “Loglog-Linear” Model

Represent a difference in double logs of probabilities on left-hand side instead. Now nice to keep signs nonnegative by inverting probabilities.

\[ \log \log (1/p_j) - \log \log (1/p_1) = \beta U_j \]

The \( \beta \) can be absorbed as \( (\frac{1}{s})^c \) even when \( c \neq 1 \) so my nonlinearized utility still fits the setting.
Alternative “Loglog-Linear” Model

Represent a difference in \textit{double logs} of probabilities on left-hand side instead. Now nice to keep signs nonnegative by inverting probabilities.

\[
\log \log (1/p_j) - \log \log (1/p_1) = \beta U_j
\]

The \( \beta \) can be absorbed as \( (\frac{1}{s})^c \) even when \( c \neq 1 \) so my nonlinearized utility still fits the setting. Then abstractly:

\[
\begin{align*}
\frac{\log (1/p_j)}{\log (1/p_1)} & = \exp (\beta U_j) =_{\text{def}} L_j \\
\log (1/p_j) & = \log (1/p_1) L_j \\
\log (p_j) & = \log (p_1) L_j \\
p_j & = p_1^{L_j}.
\end{align*}
\]

Analogy to power decay, Zipf’s Law... \textit{Proceed to demo.}